IPv6

Protokół Internetowy Następnej Generacji

Arkadiusz Miśkiewicz <*misiek@pld.org.pl*>
Polish Linux Distribution Team

10 sierpnia 1999

Streszczenie

Dokument ten prezentuje protokół IPv6. Przedstawia wymagane oprogramowanie, sposoby konfiguracji i podłączenia Linuxa do istniejących sieci wykorzystujących IPv6. Dokument zawiera także wiele wskazówek, gdzie szukać szczegółowych informacji i dodatkowego programowania.

1 Wstęp

Aktualnie jednym z istotnych problemów Internetu jest brak wolnych adresów IP. Problem ten jest częściowo rozwiązywany poprzez stosowanie translacji adresów (*NAT*). Globalnym rozwiązaniem tego problemu jest aktualnie rozwijana nowa wersja protokołu internetowego - IPv6 (znanego również jako IPng - IP Next Generation). IPv6 poza rozwiązaniem problemu braku adresów wprowadza wiele udogodnień i ulepszeń. Na dzień dzisiejszy dostępnych jest kilkanaście implementacji IPv6 (są to implementacje m.in. dla Linuxa, *BSD/KAME, Solarisa oraz Windows 9x/NT). Osobiście do pracy z IPv6 używam Linuxa oraz sporadycznie FreeBSD/KAME. Niniejszy artykuł uwzględnia głównie Linuxa.

2 Trochę teorii

Adresy IPv6 składają się z 128 bitów (dla porównana adresy IPv4 składają się tylko z 32 bitów). Łatwo jest sprawdzić, że liczba wszystkich adresów IPv6 to liczba 39 cyfrowa (dla IPv4 tylko 10 cyfrowa)! Przykładowy adres IPv6 wygląda tak: 3ffe:902:12::/48 (adres sieci). Domyślnie nie podane bity są równe „0" (np. „::" == „:0000:"). Nasz przykładowy adres podany z wykorzystaniem wszystkich bitów wyglądał będzie tak: 3ffe:0902:0012:0000:0000:0000:0000:0000/48. „/48" to długość prefiksu [1] w bitach. Taka notacja zgodna jest ze specyfikacją CIDR i dotyczy również IPv4 (RFC1518 [1], RFC1519 [2], RFC1812 [3]).

W adresach IPv6 zasięg (*scope*) adresu definiowany jest przez początkowe bity adresu i np. adresy rozpoczynające się od *fe80*: to adresy *„link-local"* - zasięg *„local"*. Poza zasięgiem local istnieją także: *host*, *site*, *global*. Po szczegółowe informacje odsyłam do RFC2373 [7] oraz do dokumentacji zawartej w pakiecie iproute2 [13] o którym będzie mowa później. Tutaj warto jedynie wspomnieć, że adresy z zasięgiem local są widoczne wyłącznie w obrębie sieci do których podpieliśmy naszego Linuxa oraz do serwerów z którymi nasz Linux ma połączenie (czy to bezpośrenie czy przy pomocy tunelu).

Istotną zaletą IPv6 jest autokonfiguracja (RFC2462 [11]). Hosty IPv6 wykorzystują między innymi protokół Neighbor Discovery (ND) pozwalający im znaleźć sąsiadujące routery i inne hosty. Dzięki ND serwery mogą śledzić, które routery lub serwery są aktywne i osiągalne, a następnie modyfikować swe tablice routingu itp. Ponadto serwery IPv6 próbują same skonfigurować swe interfejsy. Istnieją dwie metody takiej konfiguracji:

- stateless - nie wymaga żadnego konfigurowania hosta i wymaga minimalną konfigurację routerów. Metoda ta pozwala hostom na wygenerowanie własnego adresu

[1] prefiks określa ilość bitów i jest innym sposobem przedstawiania netmaski

na podstawie lokalnie dostępnych informacji i informacji rozgłaszanych przez routery [2]. Routery w tym przypadku rozgłaszają tylko prefiks sieci. Otrzymany od routera prefiks jest następnie uwzględniany podczas generowania adresów lokalnych interfejsów. Jeśli router z jakiegoś powodu nie rozgłasza odpowiednich informacji, host może wygenerować automatycznie tylko adresy *link-local* co pozwala na ograniczoną komunikację wyznaczoną zasięgiem (scope) local.

- stateful - hosty uzyskują wszelkie potrzebne informacje z serwera, który zawiera odpowiednią bazę danych. Metoda ta wykorzystuje DHCPv6.

Warto zaznaczyć, że hosty mogą wykorzystywać równocześnie obie metody do autokonfiguracji. Mechanizm obsługi IPv6 pozwala także na tworzenie dynamicznych tuneli dla pakietów IPv6 w istniejącej infrastrukturze IPv4 pod warunkiem, że adres źródłowy i docelowy pakietu to adres kompatybilny z IPv4. Wyróżniamy dwa rodzaje adresów IPv6 kompatybilnych z adresami IPv4:

- standardowe - adresy tego typu mają następujący format:

80 bitów	16 bitów	32 bity
0000.............................0000	0000	adres IPv4

- tylko IPv4 (opisujące hosty które **nie wspierają** IPv6) [3]

80 bitów	16 bitów	32 bity
0000.............................0000	FFFF	adres IPv4

Szczegóły opisane zostały w RFC1884 [4].

Do pozostałych zalet IPv6 należy zaliczyć także zmianę formatu nagłówka pakietów na nowy, pozwalający bez większych problemów dodawać w przyszłości nowe opcje bez poważnych zmian w samym nagłówku. IPv6 umożliwia także na wysyłanie datagramów zwanych jumbogramami o wielkości większej niż 65535 bajtów.

By móc wykorzystać IPv6 w obrębie dzisiejszego Internetu wykorzystującego nadal protokół IPv4 stosuje się SIT (*Simple Internet Transition*) do tunelowania pakietów IPv6 wewnątrz pakietów IPv4.

Istnieje ogólnoświatowa, wirtualna sieć bazująca na protokole IPv6. Jest to sieć 6BONE. Wirtualna dlatego, że bazuje nie na własnych, oddzielnych łączach ale wykorzystuje istniejące łącza Internetu. Niemalże wszystkie połączenia pomiędzy węzłami sieci to tunele SIT o których była mowa.

Struktura sieci składa się z głównych węzłów - pTLA (*pseudo Top Level Aggregator*), węzłów podrzędnych - pNLA (*pseudo Next Level Aggregator*) oraz podpiętych do nich

[2] narzędziem wykorzystywanym do rozgłaszania prefiksu jest radvd

[3] adresy tego typu znane są jako adresy IPv6 mapowane do IPv4 (ang. IPv4-mapped IPv6 address)

pozostałych hostów (*leaf sites*). W Polsce jedynym na dzień dzisiejszy pTLA jest ICM (*Interdyscyplinarne Centrum Modelowania Matematycznego i Komputerowego w Warszawie*), a osobą zajmującą się siecią 6bone w ICM jest Rafał Maszkowski <*rzm@icm.edu.pl*>.

3 Czego potrzeba do używania IPv6 ?

Przede wszystkim potrzebujemy jądra Linuxa najlepiej w najnowszej wersji stabilnej z serii 2.2 (dla bardziej odważnych w wersji rozwojowej z serii 2.3). Jądro należy skompilować z aktywnymi następującymi opcjami:

```
[*] Prompt for development and/or incomplete code/drivers
[*] Kernel/User netlink socket
<M> IP: tunneling
<M> The IPv6 protocol (EXPERIMENTAL)
[*] IPv6: enable EUI-64 token format
[*] IPv6: disable provider based addresses
```

Uwagi: [4] [5]

Oczywiście powyższe opcje można zarówno wkompilować w jądro jak i pozostawić w postaci ładowalnych modułów.

Kolejną rzeczą, potrzebną do kompilacji programów wykorzystujących IPv6 jest biblioteka z nowymi funkcjami opisanymi m.in. w RFC2553[12]. Właściciele glibc 2.1.1 (i nowszych) nie będą mieli żadnych problemów, gdyż ich biblioteka zawiera wszystkie [6] potrzebne funkcje. Posiadacze biblioteki libc5 mogą skorzystać z „protezy” jaką jest biblioteka libinet6 zawarta w pakiecie inet6-apps autorstwa Craiga Metza. Osobiście jednak gorąco namawiam do zaktualizowania biblioteki do najnowszej, stabilnej wersji glibc ze względu na znaczne ułatwienie przy późniejszych kompilacjach programów wykorzystujących IPv6.

Do konfiguracji IPv6 możemy wykorzystać jednen z dwóch pakietów oprogramowania - net-tools lub iproute2. Odnośniki do miejsc gdzie można znaleźć wspomniane oprogramowanie znajdziesz na końcu artykułu. Ja wykorzystuję pakiet iproute2 i o nim będzie dalej mowa.

Kompilacja iproute2 w środowisku wykorzystującym bibliotekę glibc przebiega stosunkowo bezboleśnie. W wyniku kompilacji otrzymujemy dwa programy - „ip” oraz „tc”. Pierwszy służy do konfiguracji sieci IPv4/IPv6, natomiast drugim możemy kontrolować algorytmy kolejkowania pakietów [7] (w tym także IPv6) ale to już temat na inny artykuł.

[4] niektóre narzędzia konfiguracyjne takie jak iproute2 wykorzystują „netlink” do komunikacji z jądrem

[5] opcja IP: tunneling jest konieczna jeśli chcemy tunelować pakiety IPv6 w pakietach IPv4

[6] prawie wszystkie, o czym później

[7] Linux oferuje pokaźny wachlarz algorytmów pozwalających na korzystanie z m.in. QoS (Quality Of Service)

Mając nowe, obsługujące IPv6 jądro oraz odpowiednie narzędzia możemy przystąpić do operacji jaką jest konfiguracja.

4 Konfiguracja

By sprawdzić czy IPv6 jest aktywne wystarczy wykonać komendę: „ip addr show lo”.

```
# ip addr show lo
1: lo: <LOOPBACK,UP> mtu 3924 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope global lo
    inet6 ::1/128 scope host
#
```

Jak widać IPv6 jest obecne - świadczy o tym linijka „inet6 ::1/128 scope host”. Adres „::1” jest adresem IPv6 interfejsu loopback. Jeśli powyższe polecenie nie pokazuje takiej linijki to najpewniej skompilowałeś IPv6 jako moduł. Wykonaj „modprobe ipv6” i ponownie sprawdź obecność adresu IPv6.

Działanie IPv6 można sprawdzić przy użyciu np. ping6 z pakietu iputils [8]

```
# ping6 -nc3 ::1
PING ::1(::1) from ::1 : 56 data bytes
64 bytes from ::1: icmp_seq=0 hops=64 time=0.2 ms
64 bytes from ::1: icmp_seq=1 hops=64 time=0.1 ms
64 bytes from ::1: icmp_seq=2 hops=64 time=0.1 ms

--- ::1 ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 0.1/0.1/0.2 ms
#
```

Podobnie jak w IPv4 interfejsy sieciowe (np. eth0) mogą mieć przypisanych wiele adresów IPv6. Przeglądając adresy na interfejsie ethernetowym (ip addr show eth0) zauważysz zapewne adres o którym była już mowa - link-local. Adresy te dla interfejsów ethernet generowane są automatycznie na podstawie identyfikatora interfejsu np. adresu MAC karty sieciowej. Narzędzie „iproute2” w przeciwieństwie do „ifconfig” pozwala na oglądanie wszystkich adresów na danym interfejsie. Jedną z interesujących możliwości jest dodawanie kilku adresów IPv4/IPv6 do jednego interfejsu bez stosowania aliasów (oczywiście iproute2 pozwala na stosowanie aliasów jednak w rzeczywistości żadko się tą opcję stosuje).

[8] iputils mają niewielki błąd powodujący nie działanie pinga skompilowanego egcs’em. Łata na tę dolegliwość znajduje się na *http://cvsweb.pld.org.pl/*

5 6BONE

Mając działające IPv6 możemy przystąpić do podłączania naszej maszyny do sieci 6bone. Pierwszym krokiem jest uzyskanie puli adresów IPv6. Najprościej będzie zgłosić się do Rafała Maszkowskiego <*rzm@icm.edu.pl*>. Będziesz musiał podać m.in. adres IPv4 Twojego końca tunelu SIT (czyli po prostu adres IPv4 Twojego serwera). Załóżmy, że otrzymasz sieć 3ffe:902:100::/48.

Najczęściej stosowane są dwie metody konfiguracji tuneli (w poniższych przykładach zakładam, że sieć IPv4 masz poprawnie skonfigurowaną):

- bazującą na adresach link-local.

  ```
  # echo 1 >/proc/sys/net/ipv4/ip_forward
  # echo 1 >/proc/sys/net/ipv6/conf/all/forwarding
  ```

  Pozwalamy na forwardowanie pakietów IPv4 oraz IPv6

  ```
  # ip addr add 3ffe:902:100::1/128 dev eth0
  # ip route add 3ffe:902:100::1/128 dev eth0
  ```

  Przypisujemy adres 3ffe:902:100::1/128 do interfejsu eth0 oraz ustawiamy odpowiędni routing.

  ```
  # ip tunnel add tunel mode sit local 196.34.11.5 \
  # remote 167.34.22.76 ttl 64
  # ip link set tunel up
  ```

  Tworzymy nowy tunel o nazwie „tunel” [9]. Typ tunelu to „SIT”, lokalny adres IPv4 „196.34.11.5”, natomiast adres przeciwnego końca tunelu to „167.34.22.76”. Ostatnia komenda „podnosi” interfejs naszego tunelu.

  ```
  # ip route add 3ffe::/16 via fe80::167.34.22.76 \
  # dev tunel
  ```

  Końcową operacją jest ustawienie statycznego routingu (do sieci 3ffe::0/16) poprzez router po przeciwnej stronie tunelu.

- bazującą na dynamicznych tunelach.
  Dwie pierwsze operacje, czyli pozwolenie na forwardowanie oraz przypisanie adresu do interfejsu są takie same jak w przypadku tunelu bazującego na adresach link-local. Następnie

  ```
  # ip link set sit0 up
  ```

[9] nazwa ta będzie widniała jako nazwa nowo powstałego interfejsu

Podnosimy interfejs sit0 będący interfejsem tunelu SIT (IPv6-in-IPv4).

```
# ip route add 3ffe::/16 via ::167.34.22.76 dev sit0
```

Ustawiamy statyczny routing i to już wszystko.

W powyższych przykładach adres IPv6 naszego serwera to 3ffe:902:100::1, adres IPv4 naszego serwera to 196.34.11.5, natomiast drugiej strony 167.34.22.76. Warto także nadmienić, że oba sposoby tworzenia tuneli mogą być stosowanie równocześnie.

Po tych nieskomplikowanych operacjach serwer po drugiej stronie tunelu powinien odpowiadać na pingi skierowane na jego adres IPv6 (oczywiście po drugiej stronie tunelu także należy wszystko poprawnie skonfigurować lecz to robi już osoba od której otrzymaliśmy pulę adresów IPv6). W przypadku konfigurowania tunelu z pTLA ICM adresem tym jest 3ffe:902::1. Programem przydatnym w przypadkach gdy tunel nie działa mimo teoretycznie poprawnej konfiguracji jest tcpdump pozwalający śledzić pakiety „wędrujące” przez naszego Linuxa.

6 Routing

Najczęściej w sieci 6bone stosuje się routing statyczny. Jest to rozwiązanie wystarczające w przypadku gdy posiadamy jeden tunel. Gdy liczba tuneli jest wieksza niż jeden warto zastosować routing dynamiczny bazujący na protokole BGP4+ [10] (RFC2283 [5]). Dynamiczny routing pozwala w przypadku awarii jednego z tuneli na skierowanie całego ruchu poprzez inny, istniejący tunel.

Na większości serwerów IPv6 w Polsce pracuje daemon dynamicznego routingu - **mrt** (*Multi-threaded Routing Toolkit*). Przykładowy plik konfiguracyjny („!” oznacza komentarz):

```
line vty
    login
    password haslo
    port 5674
!
enable password haslo
! 64123       - Autonomus System Number (ASN). Ze względu
!               na testowy charakter sieci 6bone można
!               wybrać dowolny ale aktualnie nie używany ASN.
router bgp 64123
! nasza podsiec
  network 3ffe:902:100::/48
! będziemy wysyłać wyłącznie zagregowane trasy (w tym
! wypadku będziemy wysyłać informację o routingu do całej
```

[10] Border Gateway Protocol z rozszerzeniami dla innych protokołów w tym IPv6

```
! naszej sieci, bez dzielenia tras na mniejsze - zmniejsza
! to obciążenie routerów BGP4+)
  aggregate-address 3ffe:902:100::/48 summary-only as-set
! będziemy informować o naszych statycznych trasach
  redistribute static
! 3ffe:902::1 - adres IPv6 na którym działa daemon
!              dynamicznego routingu naszego sąsiada
! 8664       - to numer ASN naszego sąsiada
  neighbor 3ffe:902::1 remote-as 8664
! ICM         - symbol naszego sąsiada (będzie używany
!              m.in. w logach mrt)
  neighbor 3ffe:902::1 description ICM
! bgp4+ 1     - będziemy używali BGP4+
  neighbor 3ffe:902::1 bgp4+ 1
!
! Trasy statyczne, których mrt sam nie będzie zmieniał
route 3ffe:902:100::/48 :: eth0
route 3ffe:902::1/128 fe80::167.34.22.76 tunel
```

Informacje o aktualnych trasach, które nasz mrt otrzymuje i rozgłasza możemy zobaczyć łącząc się przez telent z portem 5674 naszego serwera i wydając odpowiednią komendę. Mrt ma zaimplementowane kilkanaście komend wzorowanych na routerach Cisco opisanych w dokumentacji. Jedna z nas interesujących komend jest „show bgp"

7 DNS

Do obsługi adresów IPv6 wprowadzono nowy rekord [11] DNS jakim jest „AAAA". Jako przykład podam linijkę jaką należy dopisać do pliku opisującego naszą przykładową domenę:

```
host-ipv6 IN AAAA 3ffe:902:100::1
```

Dla odwrotnego DNSu sprawa nieco się komplikuje. Należy utworzyć delegację primary dla domeny 0.0.1.0.2.0.9.0.e.f.f.3.ip6.int, i podać adres naszego hosta:

```
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0 IN
PTR host-ipv6.nasza.domena.pl.
```

Ponieważ prefiks naszej podsieci składa się z 48 bitów (48/4=12 cyfr) to adres hosta musi sie składać z (128-48)/4=20 cyfr. Po delegację odwrotnego DNSu również musimy się zgłosić do osoby, która przydzieliła nam podsieć.

[11] Stare wersje bind-a nie obsługują tych rekordów w związku z czym konieczne staje się zaktualizowanie serwera DNS do nowszej wersji.

Sprawdzanie DNSu jest czynnością prostą - wystarczy wykonać polecenie „host -t AAAA host-ipv6.nasza.domena.pl”:

```
# host -t AAAA host-ipv6.nasza.domena.pl
host-ipv6.nasza.domena.pl IPv6 address 3ffe:902:100::1
#
```

Sprawdzanie rDNSu jest czynnością równie prostą - tym razem użyjemy narzędzia „nslookup”:

```
# nslookup -query=any 1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0\
.0.0.0.0.0.0.0.0.1.0.2.0.9.0.e.f.f.3.ip6.int
[...]
1.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.0.1.0.
2.0.9.0.e.f.f.3.ip6.int name = host-ipv6.nasza.domena.pl
[...]
```

Uwagi: [12]

8 Aplikacje

Mimo iż IPv6 nadal znajduje się w fazie eksperymentalnej to powstają aplikacje oraz łaty na już istniejące aplikacje umożliwiające im pracę z wykorzystaniem protokołu IPv6. W naszym kraju powstaje dystrybucja PLD (*Polish Linux Distribution*), mająca dostarczać użytkownikom aplikacje przystosowane do pracy w sieci IPv6. Aktualnie wszelkie podstawowe serwery usług w PLD wspierają IPv6 (www, e-poczta (MTA), ftp, ssh, telnet itd). Pełną listę aplikacji wspierających IPv6 w PLD znajdziesz na stronie http://www.pld.org.pl/

Osoby chcące pisać swe programy tak, by działały z IPv6 powinny:

- unikać stosowania struktur odnoszących się jedynie do określonej rodziny (czy to AF_INET czy AF_INET6) np. *in_addr* i *in6_addr*
- używać *getaddrinfo()* i *getnameinfo()* gdzie tylko to możliwe
- używać struktury *sockaddr_storage* mogącej przechowywać dane o adresach IPv4 i IPv6

Przystosowywanie aplikacji typu „finger” czy „telnet” jest proste, gdyż nie wymaga modyfikacji samego protokołu aplikacji. Sprawa się komplikuje jeżeli chcemy przystosować np. klienta „ftp”. W przypadku „ftp” zostały stworzone dodatkowe rozszerzenia opisane w RFC2428 ([10]), które musielibyśmy zaimplementować.

Poniżej zamieszczam przykładowy fragment kodu programu obsługujący połączenia

[12] [...] - nieistotne w tym artykule informacje o nameserwerach itp.

wyłącznie IPv4 (pierwsza wersja) oraz obsługujący połączenia IPv4 oraz IPv6 (druga wersja).

Tak wygląda typowy kod obsługujący wyłącznie IPv4:

```
const char *conhostname;
struct hostent *conhost;
struct sockaddr_in name;
int addr_len, mysock, port;

conhost = gethostbyname(conhostname);

name.sin_port = htons(port);
name.sin_family = AF_INET;
bcopy((char *)conhost->h_addr,
      (char *)&name.sin_addr,
              conhost->h_length);
mysock = socket(AF_INET, SOCK_STREAM, 0);
addr_len = sizeof(name);

connect(mysock, (struct sockaddr *)&name, addr_len);
```

Funkcja *gethostbyname()* zwraca dane dotyczące szukanego hosta, następnie funkcja *socket()* tworzy gniazdo rodziny *AF_INET*, a *connect()* inicjuje połączenie przez to gniazdo.

Odpowiadający powyższemu kod, zdolny obsługiwać zarówno IPv4 jak i IPv6 wygląda tak:

```
const char *conhostname;
struct addrinfo hints, *res, *res0;
char myport[NI_MAXSERV];
int mysock;

memset(&hints, 0, sizeof(hints));
hints.ai_family   = AF_UNSPEC;
hints.ai_socktype = SOCK_STREAM;
getaddrinfo(conhostname, myport, &hints, &res0)

for (res = res0; res; res = res->ai_next) {
        if ((mysock = socket(res->ai_family,
                             res->ai_socktype,
                             res->ai_protocol)) < 0)
                continue;
        if (connect(mysock, res->ai_addr,
                            res->ai_addrlen) < 0) {
                close(mysock);
```

```
                    mysock = -1;
                    continue;
            }
            break;
    }
    freeaddrinfo(res0);
```

Funkcja *getaddrinfo()* [13] zwraca potrzebne informacje na temat wszystkich adresów szukanego hosta. Ponieważ adresów tych może być kilka wykorzystujemy pętlę for by dla każdego z adresów wykonać odpowiednią operację. W naszym przykładzie pętla for wykonywana jest do czasu, aż funkcja *connect()* zainicjuje połączenie lub wszystkie próby zainicjowania połączenia zawiodą. *freeaddrinfo()* służy do zwalniania pamięci przydzielonej dynamicznie na potrzeby struktury „res0".

Aktualnie biblioteka systemowa glibc zawiera niemal kompletne IPv6 API co w znacznym stopniu ułatwia pisanie własnych programów zdolnych wykorzystywać IPv6. Na dzień dzisiejszy najnowsza biblioteka glibc 2.1.1 nie zawiera m.in. następujących funkcji: *getipnodebyaddr(), getipnodebyname(), rresvport_af()* jednak zostaną one zaimplementowane w wersji 2.2 biblioteki. Jak już pisałem wcześniej ważne jest to by posiadać najnowszą stabilną wersję biblioteki. Dla przykładu podam, że implementacja funkcji *getaddrinfo()* w glibc 2.1.1 ma dwa dość poważne błędy powodujące zwracanie nieprawdziwych informacji oraz zwracanie nadmiarowej ilości informacji.

Więcej informacji na temat tworzenia programów wspierających IPv6 znajdziesz w RFC2292 [6], RFC2553 [12] oraz na stronach KAME - http://www.kame.net/. Ponadto pomoc można uzyskać na jednej z list dyskusyjnych wymienionych na końcu artykułu.

9 Bezpieczeństwo

Wraz z IPv6 mają zostać wprowadzone jako standard prcedury szyfrowania oraz autentyfikacji pakietów. O szczegółach związanych z IPSec można poczytać np. w RFC2401 [8] i RFC2402 [9].

Niestety aktualnie narzędzia typu firewall IPv6 pod Linuxem dopiero się rozwijają (w Linuxie jest częściowo zaimplementowany firewall IPv6 o czym można się przekonać przeglądając źródła jądra - *linux/net/ipv6/ip6_fw.c*) w związku z czym nasz serwer może stać się nieporządaną furtką do sieci lokalnej. Ze względów bezpieczeństwa nie należy uruchamiać serwisów typu „telnetd", „finger" na publicznie dostępnych adresach IPv6. Zamiast tego możemy je uruchamiać na adresach np. *link-local*. Opcję taką umożliwia zamiennik inetd - „rlinetd". Warto także zastosować „tcp_wrappers" ze wsparciem dla IPv6. Do autentyfikacji można zastosować Kerberosa 5. Aktualnie dynamicznie rozwija

[13] ustawienie ai_family na AF_UNSPEC powoduje, że zwracane są wszelkie znany adresy szukanego hosta. Ustawienie ai_family na np. AF_INET spowodowało by, że funkcja getaddrinfo() zwróciła by adresy wyłącznie z rodziny AF_INET

się dystrybucja kerberosa o nazwie kodowej „heimdal”. Autorzy „heimdala” chcą włączyć wsparcie dla IPv6 do każdego programu wchodzącego w skład dystrybucji.

Wszelkie wymienione narzędzia można znaleźć w PLD.

10 Zakończenie

Mimo iż już działają sieci bazujące na protokole IPv6 to jednak przewiduje się, że protokół IPv4 będzie z powodzeniem panował jeszcze przez ok 5-15lat. Niemniej jednak niedawno dokonano oficjalnego przydziału adresów IPv6 dla amerykańskiego ISP z puli adresów nie testowych. Jak więc widać IPv6 zdobywa coraz większą popularność nie tylko w środowisku administratorów - eksperymentatorów.
Planuje się, że przejście na protokół IPv6 będzie odbywać się stopniowo, a sieć IPv4 i IPv6 będą przez jakiś czas współistnieć. Komunikację pomiędzy obiema sieciami mają zapewnić translatory nagłówków oraz proxy (np. SOCKS64 będący modyfikacją SOCKS5 umożliwiający komunikację hostom IPv4 z innymi hostami obsługującymi tylko IPv6 i odwrotnie). Działającą implementację takich translatorów można znaleźć w KAME (stos IPv6 dla *BSD).

Zachęcam wszystkich do eksperymentów z IPv6 oraz do przystosowywania istniejącego oprogramowania do specyfiki IPv6.

11 Dodatki

Strony WWW oraz serwisy FTP

- *http://www.6bone.net/*, *http://www.6bone.pl/*. 6bone na świecie i w Polsce
- *http://cvsweb.pld.org.pl/*, *ftp://ftp.pld.org.pl/*. Zasoby polskiego Linuxa w tym spora ilość oprogramowania współpracującego z IPv6.
- *ftp://ftp.inr.ac.ru/ip-routing/*, *ftp://ftp.icm.edu.pl/pub/Linux/iproute/*, *http://snafu.freedom.org/linux2.2/iproute-notes.html*. Narzędzia do konfiguracji sieci w tym „iproute2”, „iputils” wraz z cennymi uwagami.
- *http://www.mrtd.net/*, *http://www.zebra.org/*, *http://www.gated.org/*. Daemony dynamicznego routingu (dla IPv4 i IPv6).
- *http://www.ipv6.org/*. Użytkownicy IPv6 - forum wymiany informacji.
- *http://www.ceti.pl/~kravietz/ipv6.html*. Sprawozdanie z placu boju.
- *http://www.bieringer.de/linux/IPv6/IPv6-HOWTO/IPv6-HOWTO.html*. HOWTO opisujące uruchamianie IPv6 oraz powiązanym z nim aplikacji na Linuxie.
- *ftp://ftp.pl.kernel.org/pub/kernel/v2.2/*. Stabilne jądra Linuxa.
- *http://www(.ipv6).pld.org.pl/*. Strony Polish Linux Distribution dostępne także poprze IPv6.

- *ftp://ftp.inner.net/pub/ipv6/.* Aplikacje przystosowane do IPv6 przez Craiga Metza.
- *http://www.kame.net/.* Stos IPv6 dla systemów *BSD.
- *http://www.v6.wide.ad.jp/Papers/socks64/.* SOCKS64 - proxy IPv4<->IPv6.

Listy dyskusyjne

- 6bone - Polska. Adres listy: *6bone-pl@sunsite.icm.edu.pl*, zapisy poprzez *majordomo@sunsite.icm.edu.pl*
- 6bone. Adres listy: *6bone@isi.edu*, zapisy poprzez *majordomo@isi.edu*
- Użytkownicy IPv6. Adres listy: *users@ipv6.org*, zapisy poprzez *majordomo@ipv6.org*
- IPv6 w Linuxie. Adres listy: *linux-ipv6@inner.net*, zapisy poprzez *linux-ipv6-request@inner.net*
- Rozwój sieci w jądrze linuxa. Adres listy: *netdev@nuclecu.unam.mx*, zapisy poprzez *majordomo@nuclecu.unam.mx*

Bibliografia

[1] Y. Rekhter, T. Li, „An Architecture for IP Address Allocation with CIDR", September 1993 (RFC 1518)

[2] V. Fuller, T. Li, J. Yu, K. Varadhan, „Classless Inter-Domain Routing (CIDR)", September 1993 (RFC 1519)

[3] F. Baker, „Requirements for IP Version 4 Routers", June 1995 (RFC1812)

[4] R. Hinden, S. Deering, „IP Version 6 Addressing Architecture", December 1995 (RFC1884)

[5] T. Bates, R. Chandra, D. Katz, Y. Rekhter, „Multiprotocol Extensions for BGP-4", February 1998 (RFC2283)

[6] W. Stevens, M. Thomas, „Advanced Sockets API for IPv6", February 1998 (RFC2292)

[7] R. Hinden, S. Deering, „IP Version 6 Addressing Architecture", July 1998 (RFC 2373)

[8] R. Atkinson, „Security Architecture for the Internet Protocol", November 1998 (RFC2401)

[9] S. Kent, R. Atkinson, „IP Authentication Header", November 1998 (RFC2402)

[10] M. Allman, S. Ostermann, C. Metz, „FTP Extensions for IPv6 and NATs", September 1998 (RFC2428)

[11] S. Thomson, T. Narten, „IPv6 Stateless Address Autoconfiguration", December 1998 (RFC2462)

[12] R. Gilligan, S. Thomson, J. Bound, W. Stevens, „Basic Socket Interface Extensions for IPv6", March 1999 (RFC2553)

[13] Alexey N. Kuznetsov, „IP Command Reference", April 14, 1999